Projeto *Kiosk-IUL* (Parte 1)

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos e será composto por três partes, com o objetivo de desenvolver os diferentes aspetos da plataforma **Kiosk-IUL**. Iremos procurar minimizar as interdependências entre partes do trabalho.

Este enunciado detalha apenas as funcionalidades que devem ser implementadas na parte 1 do trabalho.

A aplicação **Kiosk-IUL** permite gerir e fazer compras num quiosque de mercearias no campus do ISCTE que está aberto 24h e é totalmente automatizado. Na aplicação **Kiosk-IUL** existem os seguintes conceitos:

1. **Utilizador**: a pessoa que compra as mercearias. Os utilizadores têm de estar registados na plataforma **Kiosk-IUL** com os seguintes dados – ID, Nome, Email, Senha, Contribuinte, Saldo (em créditos).
2. **Produto**: o item que está disponível para compra. Os produtos são caracterizados por: Nome, Categoria, Preço, *Stock* atual, *Stock* máximo.
3. **Relatório de compras**: relatório emitido com a descrição da utilização dos produtos registados no sistema, contendo o Nome de produto, Categoria de Produto, ID de utilizador, e data de compra.
4. **Relatório de *stock***: relatório emitido a pedido em que são identificados os produtos com falta de *stock*. Cada produto é identificado pelo seu nome, categoria, e número de itens em falta.

Em termos gerais, um novo utilizador regista-se na aplicação **Kiosk-IUL** fornecendo os seus dados. Ao efetuar o registo, o utilizador deve carregar o seu saldo com créditos. Após o registo, o utilizador poderá comprar produtos do quiosque quando estes estiverem em *stock*. No ato da compra, o número de itens disponível é decrementado, é descontado o preço do produto ao saldo do utilizador, e é adicionada uma linha no relatório de compras. Semanalmente, às quintas-feiras às 9h17, um procedimento automático faz o *refill* dos produtos do quiosque.

Os alunos não deverão usar o programa **echo** (não será analisado para efeitos de avaliação, a não ser que seja explicitamente indicado) para as respostas às alíneas, mas sim os scripts **`success`** (para as mensagens de sucesso) e **`error`** (para as mensagens de erro), devendo analisar os *scripts* para ver exemplos de invocação dos mesmos.

A *baseline* para o trabalho encontra-se no Tigre, na diretoria **`/home/so/trabalho-2022-2023/parte-1`**:

- **EXECUTE**, **a partir da sua diretoria local de projeto**, e sem mudar os nomes, o seguinte comando:

```
$ cp -r /home/so/trabalho-2022-2023/parte-1 .
```

Procedimento de entrega e submissão do trabalho

O trabalho de SO será realizado **individualmente**, logo sem recurso a grupos.

A entrega da Parte 1 do trabalho será realizada através da criação de **um** ficheiro ZIP cujo nome é o nº do aluno, e.g., "**a<nºaluno>-parte-1.zip**" (**ATENÇÃO:** não serão aceites ficheiros RAR, 7Z ou outro formato) onde estarão todos os ficheiros criados. Estes serão **apenas** os ficheiros de código, ou seja, na parte 1, apenas os ficheiros (***.sh *.def**). Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre **"/home/so/trabalho-2022-2023/parte-1"**, e deverá incluir nos comentários iniciais um "relatório" indicando a descrição do módulo e explicação do mesmo (poderá ser muito reduzida se o código tiver comentários bem descritivos).

Para criarem o ficheiro ZIP, usem, no Tigre, o comando **zip a<nº aluno>-parte-1.zip <ficheiros>**, por exemplo:

```
$ zip $USER-parte-1.zip *.sh *.def
```

O ficheiro ZIP deverá depois ser transferido do Tigre para a vossa área local (Windows/Linux/Mac) via SFTP, para depois ser submetido via Moodle.

Antes de submeter, por favor validem que o ficheiro ZIP não inclui diretorias ou ficheiros extra indesejados.

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do Moodle:

- Moodle da UC Sistemas Operativos, Selecionam a opção sub-menu "Quizzes & Assignments";
- Selecionem o link "Submit SO Assignment 2022-2023 Part 1";
- Dentro do formulário, selecionem o botão "Enviar trabalho" e anexem o vosso ficheiro .zip (a forma mais fácil é simplesmente fazer via "drag-and-drop") e selecionar o botão "Guardar alterações". Podem depois mais tarde re-submeter o vosso trabalho as vezes que desejarem, enquanto estiverem dentro do prazo para entrega do trabalho. Para isso, na mesma opção, pressionar o botão "Editar submissão", selecionar o ficheiro, e depois o botão "Apagar", sendo que depois pode arrastar o novo ficheiro e pressionar "Guardar alterações". **Apenas a última submissão será contabilizada**. Certifiquem-se que a submissão foi concluída, e que esta última versão tem todas as alterações que desejam entregar dado que os docentes apenas considerarão esta última submissão;
- Avisamos que a hora *deadline* acontece sempre poucos **minutos antes da meia-noite**, pelo que se urge a que os alunos não esperem por essa hora final para entregar e o façam antes, idealmente um dia antes, ou no pior dos casos, pelo menos uma hora antes. **Não serão consideradas válidas as entregas realizadas por e-mail**. Poderão testar a entrega nos dias anteriores para perceber se há algum problema com a entrega, sendo que, **apenas a última submissão conta**.

Política em caso de fraude

O trabalho corresponde ao esforço individual de cada aluno. São consideradas fraudes as seguintes situações: Trabalho parcialmente copiado, facilitar a cópia através da partilha de ficheiros, ou utilizar material alheio sem referir a fonte.

Em caso de deteção de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica da escola (ISTA) ou ao Conselho Pedagógico do ISCTE, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para deteção automática de cópias.

Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação nº 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que no âmbito do Regulamento Disciplinar de Discentes do ISCTE-IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Parte I – Shell Script (bash)

Data de entrega: 12 março de 2023

Nesta fase do trabalho, o objetivo é criar um conjunto de scripts para administração e gestão do sistema:

Atenção: Apesar do vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre "**/home/so/trabalho-2022-2023/parte-1**", assume-se que, para a sua execução, os scripts e todos os ficheiros de input e de output estarão todos **sempre** presentes na mesma diretoria, que não deve estar *hard-coded*, ou seja, os Shell scripts entregues devem correr em qualquer diretoria.

1. Script: `regista_utilizador.sh`

Este script é chamado quando um novo utilizador se conecta ou regista na aplicação, e recebe todos os dados por argumento na chamada via linha de comandos. O registo dos utilizadores é feito no ficheiro **`utilizadores.txt`**. Deve receber a informação do utilizador como argumentos, pela seguinte ordem:

`<Nome:string> <Senha:string> <Saldo a adicionar:number> [<Nr. Contribuinte:number>]`

Exemplos de invocação deste script, que deverá receber os valores pedidos passados como argumentos:

```
$ ./regista_utilizador.sh "Catarina Cruz" 12qwaszx 10
$ ./regista_utilizador.sh " Joao Baptista Goncalves" 09polkmn 20 273150235
```

Exemplo do ficheiro **`utilizadores.txt`**:

utilizadores.txt

```
1:Paulo Pereira:fsd423erddew:paulo.pereira@kiosk-iul.pt:235123532:123
2:Catarina Cruz:12qwaszx:catarina.cruz@kiosk-iul.pt:234580880:50
3:Joao Baptista Goncalves:09polkmn:joao.goncalves@kiosk-iul.pt:215654377:20
```

Com os campos inseridos como descrito acima, o script:

1.1. Valida os argumentos passados e seus formatos, **terminando o script no caso de reportar erro**:

1.1.1. Valida os argumentos passados, avaliando se são em número suficiente (mínimo 3, máximo 4); em caso de erro, dá **`error 1.1.1`** (e termina); senão dá **`success 1.1.1`**;

1.1.2. Valida se o campo **`<Nome>`** corresponde ao nome de um aluno de SO (dica: será o nome de um utilizador registado no Tigre); em caso de erro, dá **`error 1.1.2`** (e termina); senão dá **`success 1.1.2`**;

1.1.3. Valida se **`<Saldo>`** tem formato "*number*" (inteiro positivo ou `0`); se não, dá **`error 1.1.3`** e termina; caso contrário, dá **`success 1.1.3`**;

1.1.4. Valida se o formato de **`<Nr. Contribuinte>`** (apenas no caso de ter sido passado, ou seja, se tiver valor) é o de "*number*" com 9 (nove) dígitos; se não for, dá **`error 1.1.4`** (e termina); senão dá **`success 1.1.4`**.

1.2. Associa os dados passados com a base de dados de utilizadores registados:

1.2.1. Verifica se o ficheiro **`utilizadores.txt`** existe. Se o ficheiro existir, dá **`success 1.2.1`** e passa para o passo 1.2.3; se não existir, dá **`error 1.2.1`**, e não termina o script, continuando a executá-lo;

1.2.2. Cria o ficheiro **`utilizadores.txt`**; se der erro, dá **`error 1.2.2`** e termina; senão dá **`success 1.2.2`**;

1.2.3. Caso o utilizador **`<Nome>`** passado exista no ficheiro, dá **`success 1.2.3`**, e prossegue para o passo 1.3; senão dá **`error 1.2.3`**, e não termina o script, continuando a executá-lo;

1.2.4. Como o utilizador não existe no ficheiro, terá de o registar. Para isso, valida se **<Nr. Contribuinte>** (campo opcional) foi mesmo passado; se não foi, dá **error 1.2.4** e termina; senão, dá **success 1.2.4**;

1.2.5. Define o campo **<ID_utilizador>**, calculado a partir do maior **<ID_utilizador>** registado no ficheiro **utilizadores.txt**, acrescido de 1 unidade. Caso o ficheiro **utilizadores.txt** esteja vazio, dá **error 1.2.5** e define **<ID_utilizador>** = 1 (e não sai do script); caso contrário, calcula o novo **<ID_utilizador>** e dá **success 1.2.5 <ID_utilizador>** (substituindo pelo valor calculado);

1.2.6. Define o campo **<email>**, gerado a partir do **<Nome>** introduzido pelo utilizador, usando apenas o primeiro e o último nome, convertendo-os para minúsculas apenas, colocando um ponto entre os dois nomes, e domínio kiosk-iul.pt. Assim sendo, um exemplo seria " joao.goncalves@kiosk-iul.pt"; se houver algum erro na operação (e.g., o utilizador "root" não tem pelo menos 2 nomes), dá **error 1.2.6** e termina; caso contrário, dá **success 1.2.6 <email>** (substituindo pelo campo gerado);

1.2.7. Regista o utilizador numa nova linha no final do ficheiro **utilizadores.txt**, seguindo a sintaxe: **<ID_utilizador>:<Nome>:<Senha>:<email>:<Nr. Contribuinte>:<Saldo>**; se houver algum erro na operação (e.g., erro na escrita do ficheiro), dá **error 1.2.7** e termina; caso contrário, dá **success 1.2.7 <linha>** (substituindo pela linha escrita no ficheiro); prossegue para o passo 1.4.

1.3. Adiciona créditos na conta de um utilizador que existe no ficheiro **utilizadores.txt**:

1.3.1. Tendo já encontrado um "match" utilizador no ficheiro, valida se o campo **<Senha>** passado corresponde à senha registada no ficheiro; se não corresponder, dá **error 1.3.1** e termina; caso contrário, dá **success 1.3.1**;

1.3.2. Mesmo que tenha sido passado um campo **<Nr. Contribuinte>** (opcional), ignora-o. Adiciona o valor passado do campo **<Saldo a adicionar>** ao valor de créditos do Saldo registado no ficheiro para o utilizador passado, e atualiza o valor do saldo desse utilizador no ficheiro **utilizadores.txt**; se houver algum erro na operação (e.g., erro na escrita do ficheiro), dá **error 1.3.2** e termina; caso contrário, dá **success 1.3.2 <Novo Saldo>** (substituindo pelo valor do saldo calculado anteriormente).

1.4. Lista todos os utilizadores registados:

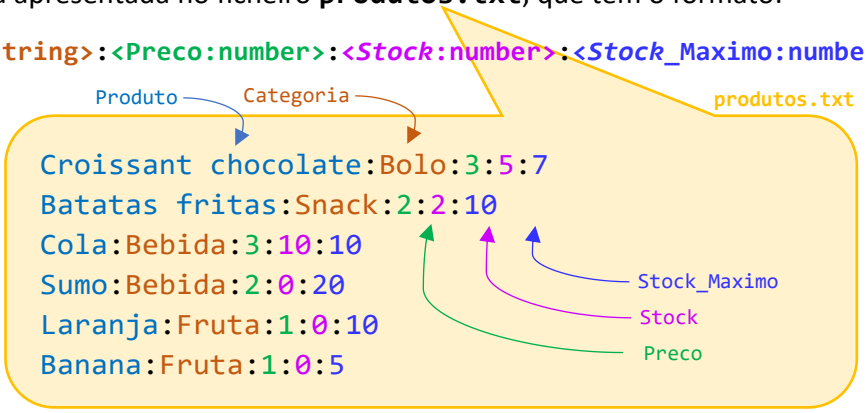
1.4.1. No final da sua execução, o script deve criar um ficheiro **saldos-ordenados.txt** igual ao que está no ficheiro **utilizadores.txt** (pode usar **echo**, **cat** ou outro), com a mesma formatação, mas com os registos ordenados por ordem decrescente de saldo de utilizadores; se houver algum erro (e.g., erro na leitura ou escrita do ficheiro), dá **error 1.4.1** e termina; caso contrário, dá **success 1.4.1**.

Este script 1 foi descrito da forma mais explícita possível, para que os alunos percebam a forma como se pretende que as perguntas deste trabalho de SO sejam respondidas, ou seja, **que TODAS as alíneas de cada script de todos os scripts (até mesmo os seguintes, onde possa apenas será dito para dar success ou error)** tenham sempre uma resposta com formato **success <alínea>** e (na maioria das vezes) também uma resposta de formato **error <alínea>**, sendo que, por vezes, a mensagem de erro implica terminar completamente o script, e noutras o comportamento esperado será continuar o script (este comportamento será especificado caso a caso). Para a validação de cada alínea, é essencial que as respostas tenham o formato especificado. Em alguns casos, poderá ser especificado que para além da mensagem de sucesso ou erro, seja passada mais informação, como é o caso da pedida em 1.3.2, 2.1.1, 2.1.2, 2.1.3, 2.2.1, 3.1.1, 3.1.2 e outros. Nesses casos, estas macros devem ser invocadas usando a sintaxe **success <alínea> <informação1> ... <informaçãoN>** ou **error <alínea> <informação1> ... <informaçãoN>**. Os scripts poderão ter outros outputs realizados com **echo** ou outras formas, mas tais outputs não serão contabilizados, a não ser quando explicitamente indicado.

2. Script: `compra.sh`

Este script permite que o utilizador compre um item da lista de produtos disponíveis. O script não recebe nenhum argumento. Os produtos estão na lista apresentada no ficheiro **`produtos.txt`**, que tem o formato:

<Produto:string>:<Categoria:string>:<Preco:number>:<*Stock*:number>:<*Stock*_Maximo:number>



Sendo **<Preco>** o preço do produto (inteiro positivo ou 0, não tem nunca um valor decimal), **<*Stock*>** o número destes produtos disponíveis na máquina de vendas, e **<*Stock*_Maximo>** o número máximo destes produtos que podem ser armazenados na máquina de vendas.
O script começa por listar no STDOUT (pode usar **`echo`**, **`cat`** ou outro) os produtos disponíveis (i.e., os que têm *stock* maior que 0), conforme o exemplo que se segue, que seria o relacionado com o ficheiro acima indicado:

```
1: Croissant chocolate: 3 EUR
2: Batatas fritas: 2 EUR
3: Cola: 3 EUR
0: Sair

Insira a sua opção: _
```

2.1. Validações e Pedido de informações interativo:

2.1.1. O script valida se os ficheiros **`produtos.txt`** e **`utilizadores.txt`** existem. Se algum não existir, dá **`error`** e termina; senão, dá **`success`**;

2.1.2. O utilizador escolhe o produto que deseja inserindo o respetivo número. Se a opção for **`0`**, dá **`success`** e termina o script. Caso o número corresponda a um produto não existente, ou um produto que não tem *stock* disponível, dá **`error`** e termina. Caso contrário, dá **`success <alínea> <Nome Produto>`**;

2.1.3. O programa pede ao utilizador o seu **<ID_utilizador>** (pode usar **`echo`**, **`cat`** ou outro):

```
Insira o ID do seu utilizador: _
```

O utilizador insere o respetivo ID de utilizador. Caso o script veja que esse número corresponda a um utilizador não registado no ficheiro **`utilizadores.txt`**, dá **`error`** e termina. Caso contrário, reporta **`success <alínea> <Nome Utilizador>`**;

2.1.4. O programa pede ao utilizador a sua **<senha>** (pode usar **`echo`**, **`cat`** ou outro):

```
Insira a senha do seu utilizador: _
```

O utilizador insere a respetiva senha. Caso o script veja que essa senha não é a registada para esse utilizador no ficheiro **`utilizadores.txt`**, dá **`error`** e termina. Caso contrário, reporta **`success`**.

2.2. Processamento da resposta:

2.2.1. Valida se o utilizador possui saldo para comprar o produto selecionado. Para isso, o script deve consultar o saldo do utilizador, definido no ficheiro **`utilizadores.txt`**, e compará-lo com o preço do produto, que está no ficheiro **`produtos.txt`**. Caso a compra não seja possível por falta de saldo, o script apresenta **`error <alínea> <preco_produto> <saldo_utilizador>`** e termina o script. Caso contrário, dá **`success <alínea> <preco_produto> <saldo_utilizador>`**;

2.2.2. Decrementa o valor do preço do produto do saldo do utilizador, e atualiza o ficheiro **`utilizadores.txt`**. Em caso de erro (por exemplo, na escrita do ficheiro), dá **`error`** e termina. Caso contrário, dá **`success`**;

2.2.3. Decrementa uma unidade ao *stock* do produto respetivo, e atualiza o ficheiro **`produtos.txt`**. Em caso de erro (por exemplo, na escrita do ficheiro), dá **`error`** e termina. Caso contrário, dá **`success`**;

2.2.4. Regista a compra no ficheiro **`relatorio_compras.txt`**, inserido uma nova linha no final deste ficheiro. Em caso de erro (por exemplo, na escrita do ficheiro), dá **`error`** e termina. Caso contrário, dá **`success`**.

O ficheiro **`relatorio_compras.txt`** tem o seguinte formato:

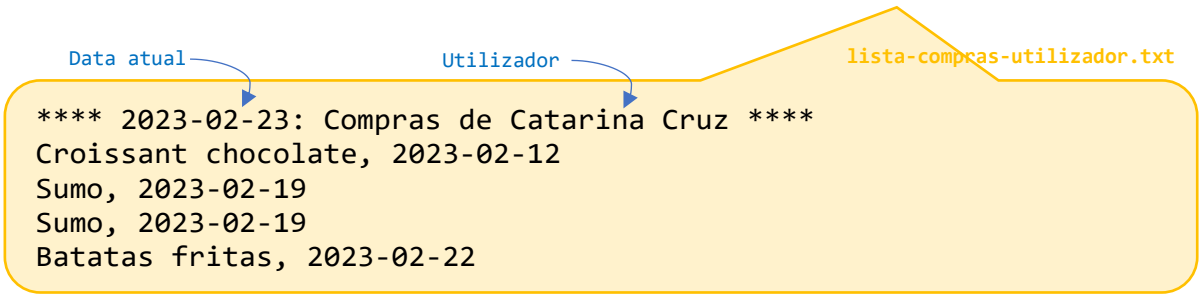
`<Produto>:<Categoria>:<ID_Utilizador>:<Data:YYYY-MM-DD>`

relatorio_compras.txt

```
Croissant chocolate:Bolo:2:2023-02-12
Sumo:Bebida:2:2023-02-19
Sumo:Bebida:2:2023-02-19
Batatas fritas:Snack:3:2023-02-21
Laranja:Fruta:1:2023-02-21
Batatas fritas:Snack:2:2023-02-22
Sumo:Bebida:3:2023-02-23
```

2.2.5. No final da sua execução, o script deve criar um ficheiro **`lista-compras-utilizador.txt`** com uma listagem (pode usar **`echo`**, **`cat`** ou outro) de todas as compras efetuadas pelo utilizador atual (e apenas esse); se houver algum erro na operação (e.g., erro na leitura do ficheiro), dá **`error`** e termina; caso contrário, dá **`success`**, e termina, não voltando a mostrar a lista de produtos inicial.

Como exemplo da listagem pretendida (dados os ficheiros de exemplo anteriores), e assumindo que, neste caso, o **`<ID_Utilizador>`** escolhido pelo utilizador é 2, o ficheiro **`lista-compras-utilizador.txt`** será:



3. Script: `refill.sh`

Este script será responsável pela reposição de *stock* de produtos. O script não recebe nenhum argumento. A reposição de *stock* é feita através da leitura do ficheiro **`reposicao.txt`**, que segue o formato:

`<Produto:string>:<Categoria:string>:<Nr_Itens_a_Adicionar:number>`

reposicao.txt

```
Croissant chocolate:Bolo:15
Batatas fritas:Snack:2
Cola:Bebida:5
Sumo:Bebida:12
Laranja:Fruta:6
Banana:Fruta:8
```

3.1. Validações:

3.1.1. O script valida se os ficheiros **`produtos.txt`** e **`reposicao.txt`** existem, dando como resultado **`error`** e terminando se algum não existir, ou **`success`** caso contrário;

3.1.2. O script valida se os produtos de **`reposicao.txt`** existem, validando se algum produto tem **`<Nr_Itens_a_Adicionar>`** que não tenha o formato "number" (inteiro positivo ou `0`), dando **`error <alínea> <nome produto>`** e terminando se tal acontecer, ou **`success`** caso contrário.

3.2. Processamento do script:

3.2.1. O script cria um ficheiro **`produtos-em-falta.txt`** onde lista os produtos em falta (para isso deve calcular a diferença entre o *stock* Máximo e o *stock* atual) dando **`error`** (e terminando) em caso de erro (por exemplo, se não conseguir escrever no ficheiro), e dando **`success`** caso contrário. O output no ficheiro (pode usar **`echo`**, **`cat`** ou outro) **`produtos-em-falta.txt`** ficará algo como:

Data atual — produtos-em-falta.txt

```
**** Produtos em falta em 2023-02-23 ****
Croissant chocolate: 2 unidades
Batatas fritas: 8 unidades
Sumo: 20 unidades
Laranja: 10 unidades
Banana: 5 unidades
```

3.2.2. O script deverá ler o ficheiro **`reposicao.txt`** e atualizar os *stocks* no ficheiro **`produtos.txt`**, adicionando às unidades do *stock* os valores de reposição para cada produto (o ficheiro **`reposicao.txt`** pode também ter outros produtos que não existem no ficheiro **`produtos.txt`**, esses elementos serão ignorados). Na atualização dos *stocks*, deverá ter atenção e garantir que o número do *stock* Máximo não é excedido na reposição de cada produto. O script dará **`error`** (e terminando) em caso de erro (por exemplo, se não conseguir escrever no ficheiro **`produtos.txt`**), dando **`success`** caso contrário.

No presente exemplo, o ficheiro resultante **`produtos.txt`** seria atualizado para:

produtos.txt

```
Croissant chocolate:Bolo:2:7:7
Batatas fritas:Snack:1.5:4:10
Cola:Bebida:2:10:10
Sumo:Bebida:1:12:20
Laranja:Fruta:0.5:6:10
Banana:Fruta:1:5:5
```

3.3. Invocação do script:

3.3.1. Altere o ficheiro **`cron.def`**, por forma a configurar o seu sistema para que o Script: `refill.sh` seja executado todas as quintas-feiras às 9h17. Nos comentários no início do ficheiro **`cron.def`**, explique a configuração realizada, e indique qual o comando que deveria utilizar para despoletar essa configuração. O ficheiro **`cron.def`** deverá ser entregue para avaliação juntamente com os outros Shell scripts (**`*.sh`**).

4. Script: `stats.sh`

Este script obtém informações sobre o sistema, afixando resultados diferentes no STDOUT consoante os argumentos passados na sua invocação.

4.1. Validações:

4.1.1. Valida os argumentos recebidos e, conforme os mesmos, o número de argumentos recebidos. Dá **`error`** e termina em caso de erro, e **`success`** caso contrário. Para facilitar, assuma que não podem ser recebidas múltiplas ordens numa mesma invocação, e.g., **`./stats.sh listar histograma`**;

4.2. Invocação do script:

4.2.1. Se receber o argumento **`listar`**, (i.e., **`./stats.sh listar`**) cria um ficheiro **`stats.txt`** onde lista (pode usar **`echo`**, **`cat`** ou outro) o nome de todos os utilizadores que já fizeram compras, por ordem decrescente de número de compras efetuadas. O script dá **`error`** e termina em caso de erro (por exemplo, se não conseguir ler algum ficheiro necessário), ou dá **`success`** caso contrário (preste atenção ao singular e plural em "compras"); o ficheiro **`stats.txt`** ficará então:

stats.txt

```
Catarina Cruz: 4 compras
Joao Baptista Goncalves: 2 compras
Paulo Pereira: 1 compra
```

4.2.2. Se receber o argumento **`popular <nr:number>`**, (e.g., **`./stats.sh popular 3`**), cria um ficheiro **`stats.txt`** onde lista (pode usar **`echo`**, **`cat`** ou outro) os **`<nr>`** (no exemplo, 3) produtos mais vendidos por ordem decrescente. O script dá **`error`** e termina em caso de erro (por exemplo, se não conseguir ler algum ficheiro necessário, ou se o argumento **`<nr>`** não tiver o formato pedido), ou dá **`success`** caso contrário (preste atenção ao singular e plural em "compras"). Em caso de empate, lista o primeiro do ficheiro; o ficheiro **`stats.txt`** ficará então:

stats.txt

```
Sumo: 3 compras
Batatas fritas: 2 compras
Croissant chocolate: 1 compra
```

4.2.3. Se receber o argumento **`histograma`**, (i.e., **`./stats.sh histograma`**), cria um ficheiro **`stats.txt`** onde lista (pode usar **`echo`**, **`cat`** ou outro) o histograma da venda dos produtos por categoria, conforme os registos no ficheiro **`relatorio_compras.txt`**. O script dá **`error`** e termina em caso de erro (por exemplo, se não conseguir ler algum ficheiro necessário), ou dá **`success`** caso contrário; o ficheiro **`stats.txt`** ficará então:

stats.txt

```
Bolo     *
Bebida   ***
Snack    **
Fruta    *
```

5. Script: `menu.sh`

Este script agrega os scripts restantes, não recebendo argumentos.

5.1. Apresentação:

5.1.1. O script apresenta (pode usar **`echo`**, **`cat`** ou outro, sem "limpar" o ecrã) um menu com as opções abaixo indicadas.

```
MENU:
1: Regista/Atualiza saldo utilizador
2: Compra produto
3: Reposição de stock
4: Estatísticas
0: Sair

Opção: _
```

5.2. Validações:

5.2.1. Aceita como input do utilizador um número, seguido de <ENTER>.Valida que a opção introduzida corresponde a uma opção válida. Se não for, dá **`error <alínea> <opcao>`** (com a opção errada escolhida), não termina o script, e volta ao ponto 5.1. Caso contrário, dá **`success <alínea> <opcao>`**;

5.2.2. Analisa a opção escolhida, e mediante cada uma delas, deverá invocar o *sub-script* correspondente descrito nos pontos 1 a 4 acima. No caso das opções **1** e **4**, este script deverá pedir interactivamente ao utilizador as informações necessárias para execução do *sub-script* correspondente, injetando as mesmas como argumentos desse *sub-script*:

5.2.2.1. Assim sendo, no caso da opção **1**, o script deverá pedir ao utilizador sucessivamente e interactivamente os dados a inserir (pode usar **`echo`**, **`cat`** ou outro):

```
MENU:
1: Regista/Atualiza saldo utilizador
2: Compra produto
3: Reposição de stock
4: Estatísticas
0: Sair

Opção: 1

Regista utilizador / Atualiza saldo utilizador:
Indique o nome do utilizador: Catarina Cruz
Indique a senha do utilizador: 12qwaszx
Para registar o utilizador, insira o NIF do utilizador:
Indique o saldo a adicionar ao utilizador: 10
_
```

Repare que, no exemplo dado acima, o utilizador não inseriu qualquer NIF, portanto, indicando que não pretende registar o utilizador, mas sim apenas adicionar saldo ao mesmo, correspondendo esta introdução assim ao primeiro exemplo de invocação via argumentos indicada em 1, que se destina apenas a adicionar 10 créditos ao saldo da utilizadora já existente Catarina Cruz, validando essa adição de créditos pela inserção da senha da utilizadora. Este script não deverá fazer qualquer validação dos dados inseridos, já que essa validação é feita no Script: `regista_utilizador.sh`. Após receber todos os dados, este script invoca o sub-Script: `regista_utilizador.sh` com os argumentos recolhidos do utilizador. Após a execução do *sub-script*, dá **`success`** e volta para o passo 5.1;

5.2.2.2. No caso da opção **2**, o script executa o *sub-script* correspondente. Após a execução do *sub-script*, dá **`success`** e volta para o passo 5.1;

5.2.2.3. No caso da opção **3**, o script executa o *sub-script* correspondente. Após a execução do *sub-script*, dá **`success`** e volta para o passo 5.1;

5.2.2.4. No caso da opção **4**, o script deverá apresentar um novo submenu (pode usar **`echo`**, **`cat`** ou outro):

```
MENU:
1: Regista/Atualiza saldo utilizador
2: Compra produto
3: Reposição de stock
4: Estatísticas
0: Sair

Opção: 4

Estatísticas:
1: Listar utilizadores que já fizeram compras
2: Listar os produtos mais vendidos
3: Histograma de vendas
0: Voltar ao menu principal

Sub-Opção: _
```

O script deverá validar que a sub-opção está entre **`0`** e **`3`**, dando **`error`** se tal não acontecer e voltando para o passo 5.1. Se o utilizador escolher a sub-opção **`0`** para voltar para o menu principal (na realidade, significa "não faz nada e volta ao início do script, mostrando novamente o menu inicial"), o script volta para o passo 5.1. Caso contrário, formata a string de invocação do sub-Script: `stats.sh`, e invoca o mesmo. Após a execução do *sub-script*, dá **`success`** e volta para o passo 5.1. Repare que, no caso particular da sub-opção **2**, há ainda um parâmetro extra a pedir ao utilizador antes de invocar o Script: `stats.sh`:

```
MENU:
1: Regista/Atualiza saldo utilizador
2: Compra produto
3: Reposição de stock
4: Estatísticas
0: Sair

Opção: 4

Estatísticas:
1: Listar utilizadores que já fizeram compras
2: Listar os produtos mais vendidos
3: Histograma de vendas
0: Voltar ao menu principal

Sub-Opção: 2

Listar os produtos mais vendidos:
Indique o número de produtos mais vendidos a listar: _
```

Só para relembrar, após a execução de cada *sub-script* invocado pelo Script: `menu.sh`, deverá sempre dar **`success`** e voltar ao ponto 5.1, apresentando novamente o menu e pedindo nova opção. Apenas a opção **`0`** (zero) permite sair deste Script: `menu.sh`. Até escolher esta opção, o menu deverá ficar em ciclo, permitindo realizar múltiplas operações iterativamente (e não recursivamente).

Anexo A: Scripts de suporte ao trabalho

Scripts **fornecidos,** com Mensagens de **sucesso**, **erro, debug** e **validação de Scripts**:

Mensagens de output com Erro (com exemplos): script `error <alínea> <argumentos>`

- Exemplo de 2.1.1: **`error <alínea> <nome ficheiro>`**

  `./error 2.1.1 utilizadores.txt`
- Exemplo de 2.2.1: **`error <alínea> <preco_produto> <saldo_utilizador>`**

  `./error 2.2.1 3 27.5`
- Exemplo de 2.1.2: **`error`**

  `./error 2.1.2`

Mensagens de output com Sucesso (com exemplos): script `success <alínea> <argumentos>`

- Exemplo de 2.1.3: **`success <alínea> <Nome Utilizador>`**

  `./success 2.1.3 "Catarina Cruz"`
- Exemplo de 5.2.1: **`success <alínea> <opcao>`**

  `./success 5.2.1 3`
- Exemplo de 5.1.1: **`success`**

  `./success 5.1.1`

Mensagens de Debug: Apesar de não ser necessário, disponibilizou-se também um Shell script para as mensagens de debug dos scripts, dado que será muito útil aos alunos: **script `debug <argumentos>`**

Um exemplo de utilização nos scripts é, **`var1=3 ; ./debug "var1:$var1"`** # mostra "@DEBUG [var1:3]"

Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando os alunos quiserem apagar as mensagens de debug, comentam a primeira linha do Shell script onde estão a trabalhar:

```
# export SHOW_DEBUG=1     ## Comment this line to remove @DEBUG statements
```

E, assim, não precisam de apagar as invocações ao script debug, mantendo os vossos scripts intocados.

Script Validador do trabalho:

Como anunciado nas aulas, está disponível para os alunos um script de validação dos trabalhos, para os alunos terem uma noção dos critérios de avaliação utilizados.

Passos para realizar a validação do vosso trabalho:

- Garantam que o vosso trabalho (i.e., os cinco scripts .sh) está localizado numa diretoria local da vossa área. Para os efeitos de exemplo para esta demonstração, assumiremos que essa diretoria terá o nome **`parte-1`** (mas poderá ser outra qualquer);
- Posicionem-se nessa diretoria **`parte-1`** da vossa área:

  $ **`cd parte-1`**
- Deem o comando $ **`pwd`** , e validem que estão mesmo na diretoria correta;
- Deem o comando $ **`ls -l`** , e confirmem que todos os ficheiros **`.sh`** do vosso trabalho estão mesmo nessa diretoria, e que esses ficheiros têm permissão de execução, e que nessa diretoria também está o link para o validador **`so-2022-trab1-validator`**;
- Agora, posicionem-se na subdiretoria do validador:

  $ **`cd so-2022-trab1-validator/`**
- E, finalmente, dentro dessa diretoria, executem o script de validação do vosso trabalho, que está na diretoria "pai" (**`..`**)

  $ **`./so-2022-trab1-validator.py ..`**
- Resta agora verificarem quais dos vossos testes "passam" (✓) e quais "chumbam" (✗);
- Façam as alterações para correção dos vossos scripts;
- Sempre que quiserem voltar a fazer nova validação, basta novamente posicionarem-se na subdiretoria **`so-2022-trab1-validator`** e correrem o script de validação como demonstrado acima.